

Java Kontrollstrukturen

➤ Kontrollstrukturen

08.11.2016

Monika Tepfenhart

1

Kontrollstrukturen

- Reihenfolge der Anweisungen innerhalb eines Methodenrumpfes vorgegeben durch
 - Reihenfolge der implementierten Anweisungen
- Kontrolliertes wiederholtes Ausführen von Anweisungen durch Kontrollstrukturen möglich
- Die wichtigsten Kontrollstrukturen sind:
 - bedingte Verzweigungen (if-else) und
 - Schleifen (for, while, do-while)

08.11.2016

Monika Tepfenhart

2

Bedingte Verzweigung

- **Bedingte Verzweigung** wird verwendet um die konkrete Stelle mit der das Programm fortgesetzt wird anhand einer Bedingung zu prüfen

➤ Struktur:

```
if (Bedingung) {  
    Anweisung1;  
}  
else {  
    Anweisung2;  
}
```

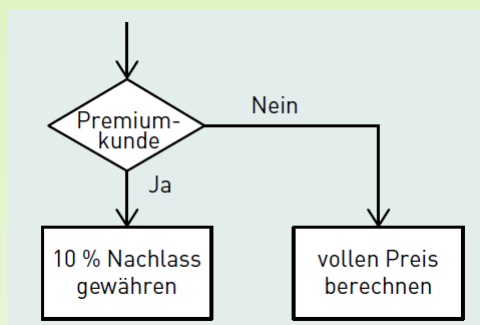
08.11.2016

Monika Tepfenhart

3

Bedingte Verzweigung

- Anwendungsfall:

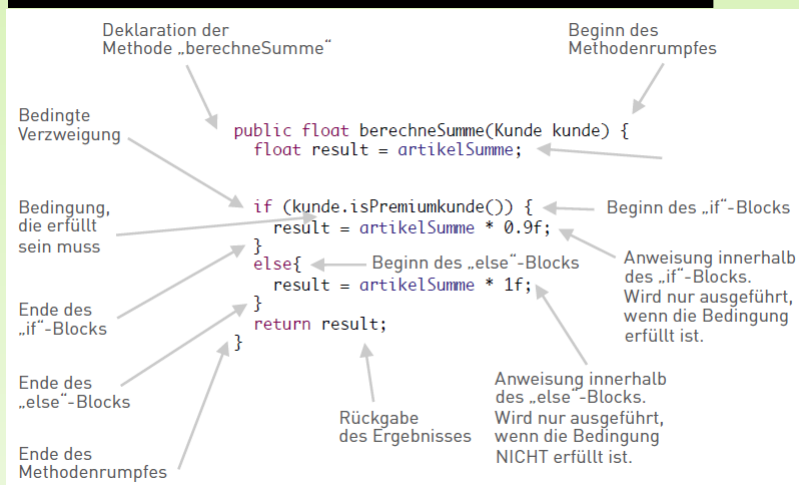


08.11.2016

Monika Tepfenhart

4

Bedingte Verzweigung



08.11.2016

Monika Tepfenhart

5

Erweiterte if-else Verzweigung

- **Erweiterten if-else Verzweigung**
 - nicht nur eine Bedingung vor dem else-Block
 - mehrere sich ausschließende Bedingungen

➤ **Struktur:**

```
if (Bedingung1) {
    Anweisung1;
}
else if (Bedingung2) {
    Anweisung2;
}
else {
    Anweisung3;
}
```

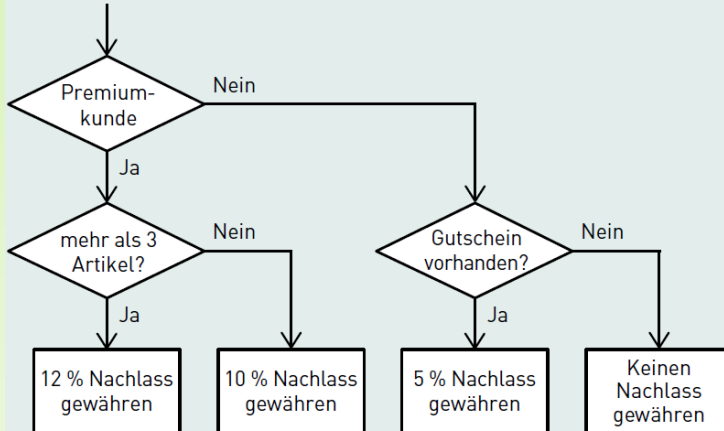
08.11.2016

Monika Tepfenhart

6

Erweiterte if-else Verzweigung

Anwendungsfall für verschachtelte Verzweigung



08.11.2016

Monika Tepfenhart

7

Erweiterte if-else Verzweigung

```
private int anzahlArtikel;  
private float artikelSumme;  
private boolean gutscheinEingeloest;  
  
public float berechneSumme(Kunde kunde) {  
    float result = artikelSumme;  
    if (kunde.isPremiumkunde()) {  
        if (anzahlArtikel > 3) {  
            result = artikelSumme * 0.88f;  
        }  
        else {  
            result = artikelSumme * 0.90f;  
        }  
    }  
    else {  
        if (gutscheinEingeloest) {  
            result = artikelSumme * 0.95f;  
        }  
    }  
    return result;  
}
```

08.11.2016

Monika Tepfenhart

8

Kontrollstrukturen

- Schleifen sind eine weitere wichtige Kontrollstruktur
 - ermöglichen die mehrfache Ausführung von gleichen Anweisungen hintereinander
 - Anzahl der Schleifendurchläufe bestimmt von einer Schleifenbedingung (auch: Laufbedingung, Abbruchbedingung)
- Drei verschiedene Schleifenarten:
 - While-Schleife,
 - Do-while-Schleife und
 - For-Schleife

08.11.2016

Monika Tepfenhart

9

while - Schleife

- **while-Schleife** prüft zuerst die Schleifenbedingung.
 - Bedingung erfüllt: Anweisungen werden ausgeführt
 - Andernfalls werden die Anweisungen übersprungen
- Nach Ausführung der Anweisungen erneute Überprüfung der Bedingung
 - Auswertung zu false: Schleife wird beendet
 - Auswertung zu true: Anweisungen der Schleife erneut durchlaufen.

08.11.2016

Monika Tepfenhart

10

while - Schleife

- Die Struktur einer **while-Schleife**:

```
while (Bedingung) {  
    Anweisungen;  
}
```

- while-Schleife auch kopfgesteuerte Schleife genannt
 - bereits vor dem ersten Ablauf wird die Bedingung geprüft
 - Auswertung zu false: komplette Schleife wird übersprungen

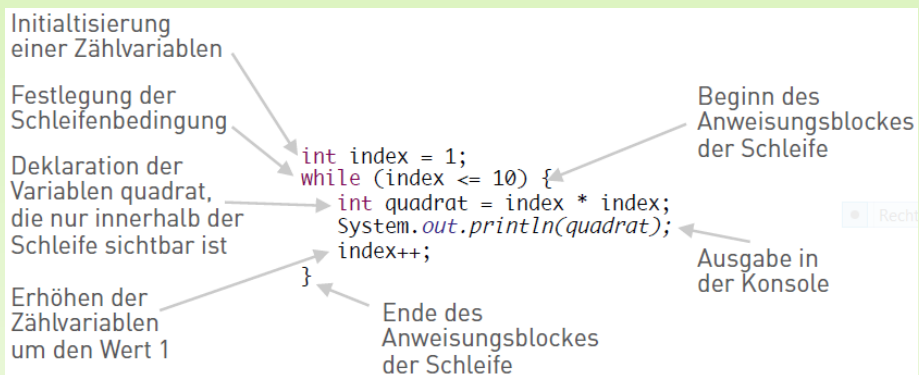
08.11.2016

Monika Tepfenhart

11

while - Schleife

Beispiel einer **while** – Schleife:



08.11.2016

Monika Tepfenhart

12

do-while - Schleife

- **do-while-Schleife** ist eine fußgesteuerte Schleife
 - Anweisungen werden mindestens einmal ausgeführt
 - Erst dann wird die Schleifenbedingung geprüft
 - Bedingung erfüllt: Anweisungen werden wiederholt
 - Bedingung nicht erfüllt: Schleife wird beendet

➤ Struktur:

```
do {  
    Anweisungen;  
} while (Bedingung)
```

08.11.2016

Monika Tepfenhart

13

do - while - Schleife

Beispiel einer **do-while – Schleife**:

Beginn des
Anweisungsblockes
der Schleife

```
int index = 1;  
do {  
    int quadrat = index * index;  
    System.out.println(quadrat);  
    index++;  
} while (index <= 10);
```

Festlegung der
Schleifenbedingung

08.11.2016

Monika Tepfenhart

14

for - Schleife

- Elemente des Schleifenkopfes bei der **for-Schleife**:
 - die Initialisierung
 - Bedingungsprüfung
 - das Ändern der Zählvariablen
- for-Schleife ist eine kopfgesteuerte Schleife
 - vor dem erstmaligen Ausführen wird Bedingung geprüft
- Struktur:

```
for (Initialisierung; Bedingung; Schleifenfortschaltung) {  
    Anweisungen;  
}
```

08.11.2016

Monika Tepfenhart

15

for - Schleife

- Ablaufschema einer **for-Schleife**:
 1. Ausführen Anweisung der Initialisierung
 2. Prüfung der Bedingung
 3. Auswerten der Bedingung
 - Wenn Bedingung true: Ausführung aller Anweisungen der for-Schleife
 - Wenn Bedingung false: Abbruch und keine Ausführung der Anweisungen
 4. Ausführen der Anweisung der Schleifenfortschaltung
 5. Weiter mit Punkt 2 (Prüfung der Bedingung)

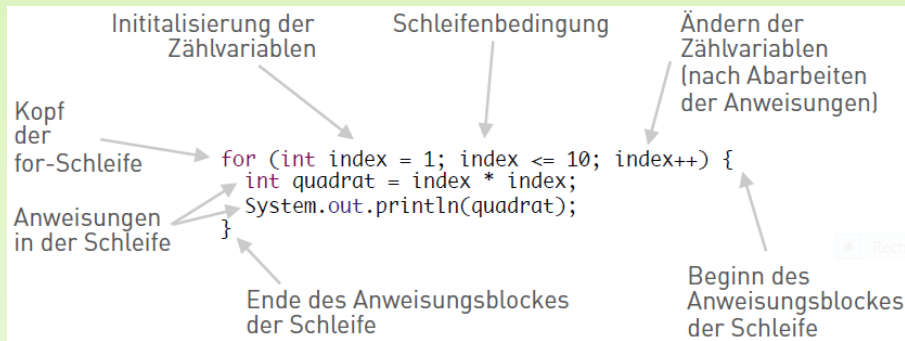
08.11.2016

Monika Tepfenhart

16

for - Schleife

Beispiel einer **for-Schleife**:



08.11.2016

Monika Tepfenhart

17

Verschachtelte Kontrollstrukturen

- Kontrollstrukturen können Kontrollstrukturen enthalten
- Schleifen können ineinander geschachtelt werden
- Codebeispiel (s. nächste Folie) mit zwei verschachtelten for-Schleifen
 - Variablen aus äußeren Kontrollstrukturen: auch in inneren Kontrollstrukturen verfügbar
 - Auf die Variable `i` kann auch innerhalb der zweiten for-Schleife zugegriffen werden
 - Anweisungen der ersten for-Schleife können nicht auf die Variable `j` der inneren for-Schleifen zugreifen

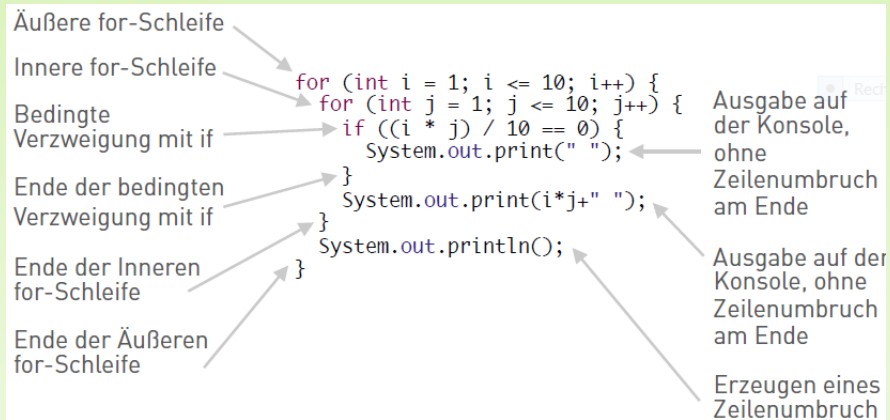
08.11.2016

Monika Tepfenhart

18

Verschachtelte for - Schleifen

Beispiel mit zwei verschachtelten **for**-Schleifen:



08.11.2016

Monika Tepfenhart

19

>>break<< und >>continue<<

➤ **break** and **continue** ist in Java mehrdeutig

- bei zwei ineinander verschachtelten Schleifen, bricht **break** nur die innere Schleife ab
 - ➔ Wie soll die äußere Schleife abgebrochen werden?
- bei zwei ineinander verschachtelten Schleifen, setzt **continue** nur die innere Schleife fort
 - ➔ Wie soll die äußere Schleife fortgesetzt werden?
- switch – Anweisung benutzt ebenfalls **break**
 - ➔ Wie soll eine Schleife, die die switch Anweisung enthält, abgebrochen werden?

08.11.2016

Monika Tepfenhart

20

>>break<<

- **break** bei verschachtelten for - Schleifen

5	1	5
5	2	10
5	3	15
5	4	20
5	5	25

```
public static void demoBreak() {  
    int product = 0;  
    for (int i = 0; i < 100; i++) {  
        for (int j = 0; j < 100; j++) {  
            product = i * j;  
            System.out.printf("%4d %4d %6d %n", i, j, product);  
            if (product == 25) {  
                System.out.println("BREAK");  
                break;  
            }  
        }  
    }  
}
```

BREAK

6	0	0
6	1	6
6	2	12
6	3	18

08.11.2016

Monika Tepfenhart

21

>>break<<

- **break** bei switch Anweisung innerhalb einer for - Schleifen

```
public static void demoSwitchBreak(){  
    for (int i = 1; i < 10; i++) {  
        switch (i) {  
            case 1:  
                System.out.printf("i:%2d %n", i); break;  
            case 2:  
                System.out.printf("i:%2d    i²: %2d %n", i, i*i); break;  
            case 3:  
                System.out.printf("i:%2d    i³: %2d %n", i, i*i*i); break;  
            default: System.out.printf("i:%2d ist groesser als drei %n", i); break;  
        }  
    }  
}
```

i: 1
i: 2 i²: 4
i: 3 i³: 27
i: 4 ist groesser als drei
i: 5 ist groesser als drei
i: 6 ist groesser als drei
i: 7 ist groesser als drei
i: 8 ist groesser als drei
i: 9 ist groesser als drei

08.11.2016

Monika Tepfenhart

22

>>continue<<

- **continue** bei zwei ineinander verschachtelten for - Schleifen

```
public static void demoContinue(){
    int product = 0;
    for (int i = 0; i < 100; i++) {
        for (int j = 0; j < 100; j++) {
            product = i * j;
            if (product == 25) {
                System.out.println("CONTINUE");
                continue;
            }
            System.out.printf("%4d %4d %6d %n", i, j, product);
        }
    }
}
```

5	0	0
5	1	5
5	2	10
5	3	15
5	4	20
CONTINUE		
5	6	30
5	7	35
5	8	40
5	9	45
5	10	50

08.11.2016

Monika Tepfenhart

23

>>break<< und >>continue<< mit Marken

- Definition einer **Marke** (engl. **Label**)
- Bezeichner der Marke wird mit Doppelpunkt abgeschlossen vor eine Anweisung gesetzt
- Die break Anweisung wird mit der Marke markiert

08.11.2016

Monika Tepfenhart

24

>>break<< mit einer Marke für Schleifen

➤ break mit Marke >>outer<< bei äußerer for - Schleife

```
public static void demoBreakLabel() {  
    int product = 0;  
    outer:  
    for (int i = 0; i < 100; i++) {  
        inner:  
        for (int j = 0; j < 100; j++) {  
            product = i * j;  
            System.out.printf("%4d %4d %6d %n", i, j, product);  
            if (product == 25) {  
                System.out.println("BREAK");  
                break outer;  
            }  
        }  
    }  
    System.out.println("Ende der Methode demoBreakLabel()");  
}
```

08.11.2016

Monika Tepfenhart

25

>>break<< mit einer Marke für Schleifen

➤ break mit Marke >>outer<< bei äußerer for - Schleife

```
1  19  19  
1  20  20  
1  21  21  
1  22  22  
1  23  23  
1  24  24  
1  25  25  
BREAK  
Ende der Methode demoBreakLabel()
```

08.11.2016

Monika Tepfenhart

26

>>break<< mit einer Marke für Schleifen

➤ break mit Marke >>inner<< bei innerer for - Schleife

```
public static void demoBreakLabel() {  
    int product = 0;  
outer:  
    for (int i = 0; i < 100; i++) {  
        inner:  
        for (int j = 0; j < 100; j++) {  
            product = i * j;  
            System.out.printf("%4d %4d %6d %n", i, j, product);  
            if (product == 25) {  
                System.out.println("BREAK");  
                break inner;  
            }  
        }  
    }  
    System.out.println("Ende der Methode demoBreakLabel()");  
}
```

08.11.2016

Monika Tepfenhart

27

>>break<< mit einer Marke für Schleifen

➤ break mit Marke >>inner<< bei innerer for - Schleife

0	0	0
5	1	5
5	2	10
5	3	15
5	4	20
5	5	25
BREAK		
6	0	0
6	1	6
6	2	12
6	3	18
6	4	24
6	5	30

08.11.2016

Monika Tepfenhart

28

>>break<<

➤ **break mit Marke >>forloop<<** bei switch - Anweisung innerhalb einer for - Schleifen

```
i: 1
i: 2    i²: 4
```

```
public static void demoSwitchBreakLabel(){
    forloop:
    for (int i = 1; i < 10; i++) {
        switch (i) {
            case 1:
                System.out.printf("i:%2d %n", i); break;
            case 2:
                System.out.printf("i:%2d    i²: %2d %n", i, i*i); break forloop;
            case 3:
                System.out.printf("i:%2d    i³: %2d %n", i, i*i*i); break;
            default: System.out.printf("i:%2d ist groesser als drei %n", i); break;
        }
    }
}
```

08.11.2016

Monika Tepfenhart

29

>>continue<<

➤ **continue mit Marke >>outer<<** bei zwei ineinander verschachtelten for - Schleifen

```
public static void demoContinueLabel(){
    int product = 0;
    outer:
    for (int i = 0; i < 100; i++) {
        inner:
        for (int j = 0; j < 100; j++) {
            product = i * j;
            if (product == 25) {
                System.out.println("CONTINUE OUTER");
                continue outer;
            }
            System.out.printf("%4d %4d %6d %n", i, j, product);
        }
    }
}
```

5	0	0
5	1	5
5	2	10
5	3	15
5	4	20
CONTINUE OUTER		
6	0	0
6	1	6
6	2	12
6	3	18
6	4	24
6	5	30

08.11.2016

Monika Tepfenhart

30

>>continue<<

➤ **continue mit Marke >>inner<<** bei zwei ineinander verschachtelten for - Schleifen

```
public static void demoContinueLabel(){
    int product = 0;
    outer:
    for (int i = 0; i < 100; i++) {
        inner:
        for (int j = 0; j < 100; j++) {
            product = i * j;
            if (product == 25) {
                System.out.println("CONTINUE INNER");
                continue inner;
            }
            System.out.printf("%4d %4d %6d %n", i, j, product);
        }
    }
}
```

5	0	0
5	1	5
5	2	10
5	3	15
5	4	20
CONTINUE INNER		
5	6	30
5	7	35
5	8	40
5	9	45
5	10	50

08.11.2016

Monika Tepfenhart

31